

Elsevier L^AT_EX template^{*}

Elsevier¹

Radarweg 29, Amsterdam

Elsevier Inc^{a,b}, Global Customer Service^{b,}*

^a1600 John F Kennedy Boulevard, Philadelphia

^b360 Park Avenue South, New York

Abstract

This template helps you to create a properly formatted L^AT_EX manuscript.

Keywords: `elsarticle.cls`, L^AT_EX, Elsevier, template

2010 MSC: 00-01, 99-00

1. Introduction

Since the dawn of internet and world wide web, humanity has witnessed a degree of connection beyond reckoning. The proliferation of digital devices pervaded with various applications that account for almost all aspect of human-
ity, have created cyber communities that constantly mutate [1]; [2]. In a world
where we have network infrastructures that can support up to 250Mbps of data
transmission, and smart phones and IOT devices that can have processing power
of up to 3 Ghz, data becomes ubiquitous, the quantum that lays the foundation
of the nexus [3].

According to InternetLiveStates.com [4], only in one second, there are 9,878
tweets sent, 1,138 instagram photos uploaded, 3,117,720 emails sent, 99,738
Google searches made, and 94,144 Youtube videos viewed. That is, if it has

^{*}Fully documented templates are available in the elsarticle package on CTAN.

^{*}Corresponding author

Email address: `support@elsevier.com` (Global Customer Service)

URL: `www.elsevier.com` (Elsevier Inc)

¹Since 1880.

taken 5 second the read the preceding paragraph, during that time, 15,588,600 emails are sent.

15 Driven by the ambition to harness the power of this deluge of data, the term 'Big Data' (BD) was coined [5]. BD initially emerged to address the challenges associated with various characteristics of data such as velocity, variety, volume and variability [2]. BD is the practice of extracting patterns, theories, and predictions from a large set of structured, semi-structured, and unstructured
20 data for the purposes of business competitive advantage [6]; [7]. BD is a game-changing innovation, heralding the dawn of a new data-oriented industry.

Nonetheless, BD is not a magical wand that can enchant any business process. While a lot of opportunities exist in BD, subsuming an emergent and rather high-impacting technology like BD to current state of affairs in organi-
25 zations, is a daunting task. According to recent survey from Databricks, only 13% of the organizations excel at delivering on their data strategy [8]. Another survey by NewVantage Partners indicated that only 24% organization have successfully gone data-driven [9]. This survey also states that only 30% of organizations have a well established strategy for their big data endeavour. In
30 addition, surveys from McKinsey & Company ([10]) and Gartner ([11]) further support these numbers, which illuminates on the scarcity of successful big data implementations in the industry.

Among the challenges of data adoption perhaps the most highlighted are 'data engineering complexities', 'big data architecture', 'rapid technology change',
35 'lack of sufficient skilled data engineers', and 'organization's cultural challenges of becoming data-driven' [2];[12]. This focus of this study is on data engineering complexities and in specific big data architecture.

In the past, organization relied on a few technology giants to provide infrastructure and tools necessary for big data, while today there's a plethora of
40 choice from hundreds of providers covering different aspect of data ecosystem from ingestion, to logging, to stream processing, and to visualization [9]. Companies are tending more and more towards Cloud-native architectures for cost reduction, improved efficiency and new roles have been introduced such as chief

analytics officer (CAOs) and chief data officers (CDOs) to channel the organizational big data capabilities toward business value and competitive advantage [13].

So how can one embark on this rather sophisticated journey? what can be a good logical approach to absorb the ever-increasing complexity of big data systems? how can organizations build different stacks to handle data for various workloads such as machine learning (ML), business analytics, data engineering, and streaming?

We suggest that majority of the challenge discussed starts with data architecture [1]; [3]. The data ingestion, processing and consumption of different data workloads vary, and sometimes they don't go well together. A company that enacted a data lake and a data warehouse and tries to account for both ecosystems, can be dealing with immense complexity, which in turns impact data teams, which in turn can hinder innovation, create barriers and result in monumental lost.

Development and deployment of an efficacious big data system is only the beginning of a big data journey. As data sources increase, variety of data increases, number of data consumers increase, the data store gets confuscated, and this can introduce threats for scalability and maintainability of the system. This also implies that only a handful of hyper-specialized data engineers would understand the system internals, creating silos, and potential miscommunication.

Majority of these systems are developed on-premise as ad-hoc complicated solutions that do not adhere to the practices of software engineering and software architecture [14]; [15]. As the ecosystem grows and new technologies and data processing techniques are introduced, the software architect will have a harder time to come up with a solution that address the problem requirements.

This can potentially create grounds for an immature architecture that results in solutions that are hard to scale, hard to maintain, and raise high-entry blockades [3]. Since the approach of ad-hoc design to big data system development is not desirable and may leave many architects and data engineers in the dark,

75 novel data architectures that are designed specifically for BD are required. To
contribute to this goal, we explore the notion of reference architectures (RAs)
and present a distributed domain-driven software RA for big data systems.

2. Why reference architecture?

To justify why we have chosen reference architectures as the suitable artefact,
80 first we have to clarify two assumptions;

1. having a sound software architecture is essential to the successful devel-
opment and maintenance of software systems
2. there exist a sufficient body of knowledge in the field of software architec-
ture to support the development of an effective RA

85 One of the focal tenets of software architecture is that every system is devel-
oped to satisfy a business objective, and that the architecture of the system is a
bridge between abstract business goals to concrete final solutions [16]. While the
journey of big data can be quite challenging, the good news is that a software
RA can be designed, analyzed and documented incorporating best practices,
90 known techniques, and patterns that will support the achievement of the busi-
ness goals. In this way, the complexity can be absorbed, and made tractable.

Practitioners of complex systems, software engineers, and system designers
have been frequently using reference architectures to have a collective under-
standing of system components, functionalities, data-flows and patterns which
95 shape the overall qualities of system and help further adjust it to the business
objectives [17]; [18]. There is a fair amount of literature on reference architec-
tures, and whereas different authors definition may vary, they all share the same
tenets.

A reference architecture is amalgamation of architectural patterns, stan-
100 dards, software engineering techniques that bridge the problem domain to a
class of solutions. This artefact can be partially or completely instantiated and
prototyped in a particular business context together with other supporting arte-

fact to enable its use. RAs are often created from previous RAs and architecture [1].

105 The usage of RAs for the development of complex systems is not new. In software product line (SPL) development, RAs are generic artifacts that are configured and instantiated for a particular domain of systems [19]. In software engineering, major IT giants like IBM has referred to RAs as the 'best of best practices' to address unique and complex system development challenges [17].

110 Based on the premises discussed and taking all into consideration, RAs can facilitate the issues of big data architecture and data engineering because of the following reasons;

1. RAs can promote adherence to best practice, standards, specifications and patterns
- 115 2. RAs can endow the data architecture team with openness and increase operability, incorporating architectural patterns that ensue desirable pre-defined quality attributes
3. RAs can be the best initial start to the big data journey, capturing design issues when they are still cheap
- 120 4. RAs can bring different stakeholders on the same table and help achieve consensus around major technological constructs
5. RAs can be effective in identifying and addressing cross-cutting concerns
6. RAs can serve as the organizational memory around design decisions, enlightening next subsequent decisions
- 125 7. RAs can act as a summary and blueprint in the portfolio of software engineers and architect, resulting in better dissemination of knowledge

3. Research Methodology

There are a few studies that have addressed the systematic development of reference architectures. Cloutier et al [17] present a high-level model for
130 RA development through collection of contemporary information and capturing the essence of architectural advancements. In another effort, PuLSE-DSSA

is proposed by Bayer et al. [20] in the context of product line development and domain engineering. PulSE-DSSA emphasizes on capturing the existing architectural knowledge. Stricker et al. [21] propose a pattern-based approach
135 for creating an RA. This study revolves around software engineering patterns motivated by the work of Gamma et al [22]; proposing a structural approach that includes three layers of patterns with well-defined hierarchical relationships. Nakagawa, Martins, Felizardo, and Maldonado [23] propose an approach to RA design outside of product line management context that is concentrated
140 towards aspect-oriented systems.

Galster and Avgeriou [24] propose an empirically grounded reference architecture based on two main facets; Existing RAs in practice and available literature on RAs. Along the same vein, Nakagawa et al [25] presented ProSA-RA which is a 4 phase methodology that unlike many other methodologies do
145 provide a more comprehensive instructions on RA evaluation. In addition, this methodology benefits from an ecosystem of complementary constructs that aid in RA design and evaluation such as RAModel [26] and a framework for evaluation of RAs (FERA) [27]. In a recent study, Derras et al. [28] propose a schema of practical RA development in the context of software product line and domain
150 engineering. This study is based on capturing knowledge from architectures in practice with attention to variability, configurability and product line development. The findings provide a four-phase process to develop quality driven reference architectures. This approach is influenced by ISO/IEC 26550 [29].

By analysis and study of all these approaches for design and development of
155 RAs, a common pattern has been witnessed. Whereas some of them are more recent and some belong to years ago, there are commonalities that has been observed. All these approaches are grounded on three main pillars, 1) Existing RAs 2) RAs in literature 3) Architectures in practice. Taking this into consideration and by analyzing the results of the systematic literature review conducted
160 by Ataei et al [1] we found 'Empirically-grounded reference architectures' proposed by Galster and Avgeriou [24], a suitable methodology, because firstly it's been adopted by many studies, and secondly it's comparatively in-line with the

nature of our study.

Nevertheless, we did not fully adopt this methodology and rather customized
165 to the needs of this particular research. This is due to some inherent limitations
that has been witnessed with the methodology. For instance we could not find
a comprehensive guideline on how to identify data sources and how it could
be categorized and synthesized into the creation of the RA in the third step
of the methodology, therefore we employed the Nakagawa’s information source
170 investigation guidelines and the overall idea of the RAModel. Another limitation
we’ve faced was with evaluation of the RA. As evaluation, second to a sound
research methodology is one of the key elements of any good design science
research, we had to look for a stronger and more systematic evaluation approach
than what was discussed in ‘empirically grounded RAs’ methodology. For this
175 purpose, and inspired by the works of Angelov et al [30]; [31], we first created
an prototype of the RA in practice, and then used ‘The architecture tradeoff
analysis method’ (ATAM) [32] to evaluate the artefact.

This research methodology is constituent of 6 phases which are respectively;
1) Decision on the type of the RA 2) Design strategy 3) Empirical acquisition
180 of data 4) Construction of the RA 5) Enable RA with variability 6) Evaluation
of the RA. The phrase ‘empirically grounded’ refers to two major elements;
firstly the reference architecture should be grounded in well-established and
proven principles; secondly, the reference architecture should be evaluated for
applicability and validity. These don’t only belong to Galster and Avgeriou
185 methodology, and other researchers such as Cloutier [17] and Derras et al [19]
have promoted the same ideas.

It is worth mentioning that this methodology is iterative, meaning that the
results gained from the evaluation phase (6th phase) determines the subsequent
iterations until the design reaches saturation.

190 3.1. Step1: Decision on type of the RA

Precursor to any effective RA development, is the decision on type of it. The
type of the RA is significant, as it illuminates on information to be collected

and the construction of the RA in later phases. The selection on the type of RA for the purposes of this study is based on two dimensions; the classification
195 framework proposed by Angelov et al. [33] and the usage context [34].

Based on the classification framework proposed by Angelov et al. [33], five types of RA are defined. This framework has been developed with the goal of supporting analysis of RAs with regards to context, goal, and the architecture specification/design relationships. It is based on 3 major dimensions namely
200 context, goals, and design, each having their own corresponding sub-dimensions. These dimensions and sub-dimensions are derived by the means of interrogatives (the usage of interrogatives is a well-established practice for problem analysis (the usage of interrogatives is a well-established practice for problem analysis)).

The interrogatives ‘When’, ‘Where’, and ‘Who’ have been used to address
205 the ‘context’, ‘Why’ has been used to address ‘goal’, and ‘How’ and ‘What’ have been used to address ‘design’ dimension. The outcome of the study categorizes RAs in two major groups; 1) standardization RAs and 2) Facilitation RAs. This framework has been chosen because it is completely in-line with the purposes of this study and aims to demarcate a clear domain for the RA to be developed.
210 The comprehensive classification of the RAs with examples in practice illuminates on how different RAs are playing roles in the industry and how they are classified. This brings clarity on what should be developed and what boundaries should be drawn.

By reading the results of the recent SLR conducted by Ataei et al on BD
215 RAs [1], we’ve added more examples of the RAs on top of what was provided by Angelov [33], and provided the following updated list of RA classifications with examples;

1. Standardization RAs

- (a) Type 1: classical, standardization architectures designed to be im-
220 plemented in multiple organizations. Examples are:
 - i. WRM [35]
 - ii. OSI RM [36]

- iii. OATH [37]
 - iv. COBRA [38]
 - 225 v. Neomycelia [3]
 - vi. Kappa [39]
 - vii. Bolster [15]
- (b) Type 2: classical, standardization architectures designed to be implemented in a single organization
- 230 i. Fortis Bank Reference Software Architecture [?]
- 2. Facilitation RAs
- (a) Type 3: classical, facilitation reference architectures for multiple organizations designed by a software organization in cooperation with user organizations
- 235 i. Microsoft Application Architecture for .Net [40]
- ii. IBM PanDOORA
- iii. OATH [37]
- iv. COBRA [38]
- (b) Type 4: classical, facilitation architectures designed to be implemented in a single organization
- 240 i. Achmea Software Reference Architecture [41]
- ii. ABN-AMRO Web Application Architecture [42]
- (c) Type 5: preliminary, facilitation architectures designed to be implemented in multiple organizations
- 245 i. ERA [34]
- ii. AHA [43]
- iii. eSRA [44]

The domain driven distributed BD RA chosen for the purposes of this study pursues two major goals; 1) enabling and support the development and data engineering of big data systems 2) concurrently ensuring that interoperability between different heterogeneous components of the big data system is established. Therefore, the outcome artefact will be a BD RA that is a classical standardization RA designed to be implemented in multiple organizations.

3.2. Step2: Selection of Design Strategy

255 Angelov et al [30] and Galster et al[24] have both presented that RAs can have two major design strategies to them; 1) RAs that are designed from scratch (practice driven), 2) RAs that are based on other RAs (research driven). Designing RAs from scratch is rare, and usually takes place in an emergent domain that have not perceived a lot of attention. On the other hand, most RAs today
260 are the amalgamation of a priori concrete architectures, models, patterns, best practices, and RAs, that together provide a compelling artefact for a class of problems.

RAs developed from scratch tend to create more prescriptive theories, whereas RAs developed based on available body of knowledge tends to provide with more
265 descriptive design theories. The RA designed for the purposes of this study is a research-based RA based on existing RAs, concrete architectures, and best practices.

3.3. Step 3: Empirical Acquisition of Data

As aforementioned, due to the limitation witnessed by this research method-
270 ology, we have augmented this phase, and increase the systematicity and transparency of data collection and synthesis through various academic methods such as systematic literature review or SLR.

This phase is made up of three major undertakings; 1) identification of data sources; 2) capturing data sources; 3) synthesis of data sources.

275 3.3.1. Identification of data sources

To identify suitable data sources, we've employed the first step of ProSA-RA methodology, 'information source investigation'. This step is an endeavour to capture focal and ancillary knowledge and theories that revolve around the target domain, and lay the ground of RA development.

280 To unearth the architectural quanta, and to highlight gradations between various approaches to BD system development, we've selected most relevant sources as the followings;

1. **Practice-led conferences:** given that majority of recent advancements for emerging technologies such as microservices architecture [45];[46] [47] and big data are coming from virtually hosted practice-led conferences, we've chosen some of the best conferences hold world-wide for the purposes of data collection. These conferences are 1) Qcon [48] 2) State of Data Mesh by ThoughtWorks [49] 3) Worldwide Software Architecture Summit'21 [50] and 4) Kafka Summit Europe 2021 [51]. Our objective was to capture the frontiers of software architecture and emerging approaches currently being practiced in IT giants such as Google, Facebook and Netflix. Among all the speech in these conferences, we looked for topics that entailed the keywords 'emergent software architecture trends', 'distributed software architecture', and 'big data software architecture'. We used the software Nvivo to code the transcripts from the conference videos. We used the aforementioned keywords as the codes and associated different texts, summative, essence-capturing sentences, evocative attributes to them. During this process, a new theme 'domain driven design' emerged. We added that into the list of codes as well.
2. **Publications:** in order to capture evidence from the body of knowledge, we conducted a systematic literature review (SLR), following the guidelines of PRISMA presented by Moher et all [52]. The main objective of this SLR was to highlight common architectural constructs found among all the BD RAs. This SLR is build on top of our recent work [1] that covered all the RAs by 2020. The initial SLR included IEEE Explore, ScienceDirect, SpringerLink, ACM library, MIS Quarterly, Elsevier, AISel as well as citation databases such as Scopus, Web of Science, Google Scholar, and Research Gate. The SLR search keywords used were 'Big Data Reference Architectures', 'Reference Architectures in the domain of Big Data', and 'Reference Architectures and Big Data'. We followed the exact methodology, but this time for the years 2021 and 2022. Our aim was to find out if there has been any new BD RA published during the years mentioned.

By the result of this SLR, we've found 3 more BD RAs ([3]; [53]; [54])
 and we've added two new standards ([55]; [56]) to further solidify our
 study. Converging these new SLR with the old, covering the years 2010-
 2022, we've pooled 89 literature in the primary phase, and another 10 by
 snowballing and citation searching. These 99 literature then went through
 our inclusion, exclusion and quality criteria. These criteria is as blow;

- Inclusion criteria:

- (a) studies that entailed real-world scenarios or tend to solve a problem in practice
- (b) qualitative or quantitative researches that intended to solve the industry gaps in big data system development and architecture
- (c) studies that had strong evaluations, preferably those that created the artefact in an actual organizational setup
- (d) explores the concept of RAs
- (e) provides or builds up on thorough discussion on BD RAs, limitations, drivers, and the overall ecosystem
- (f) is recent, within the years specified
- (g) is a conference paper, journal paper, book, book chapter, white paper, dissertation or thesis

- Exclusion criteria:

- (a) the study is not well evaluated or practice driven
- (b) is a duplicate
- (c) the study is not in-line with research objectives
- (d) not written in English
- (e) provides a poor quality or misleading technical information

- EQuality assessment:

- (a) is the study rich in terms of relevance to practice?
- (b) does the study create related design/design science or kernel theories?
- (c) does the study entail sufficient data?

(d) does the study discuss the recent trends in BD domain?

(e) is the study based on primary data and is internationally focused?

3.3.2. Data Synthesis

After pooling the studies, we removed 11 studies before screening because either they were duplicates or they were not in English. The remaining 78 studies went through screen, in which, 2 studies excluded based on the exclusion criteria. From there on, 76 studies have been assessed for eligibility based on the quality framework and the inclusion criteria. The result of this process handed over 67 studies from this branch. From the other branch, 10 records identified through citation searching. These reports have been assessed through the same quality framework, inclusion and exclusion criteria, which yielded 5 studies from this stream. Together 68 studies pooled for this SLR as depicted in figure 1.

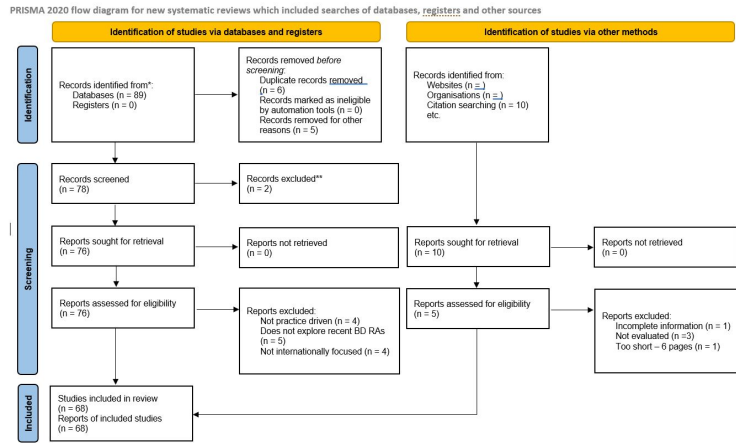


Figure 1: PRISMA flowchart

These 68 studies are comprising of journal papers, conference papers, book chapters, tech reports, tech surveys white papers, standards, master thesis, and PhD dissertations. Out of the pool of these studies, 39.4% are from IEEE Explore, 4.4% are from ScienceDirect, 23.5% are from Springerlink, 13.2% are from ACM, and 29.4% are from other sources such as citation search, Google Scholar

and Research Gate. 30 journal articles, 14 conference papers, 6 whitepapers, 2 ISO standards, 14 book chapters, and 2 postgraduate studies have been selected. 26% of these studies are from the year 2010-2013, 33% are from the years
 365 2013-2015, and 51% are from the years 2016-2022. These stats are portrayed in figure 2.

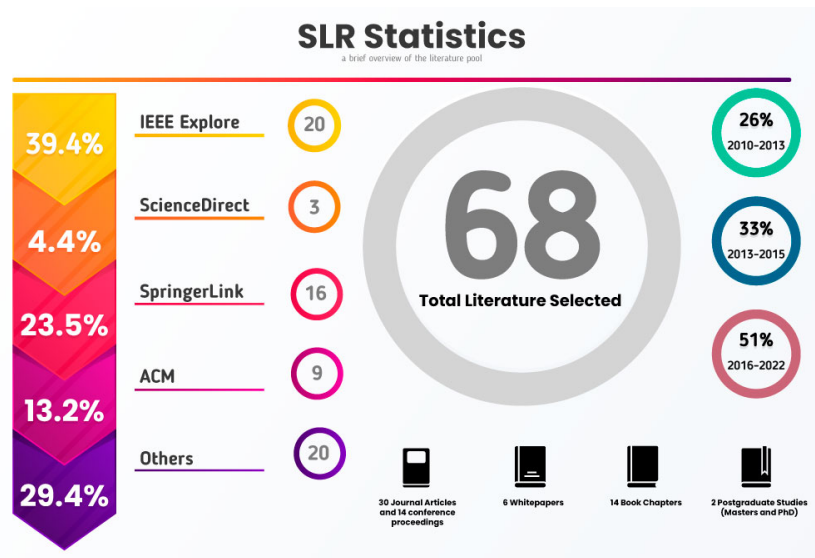


Figure 2: SLR statistics

By this stage, the research objective is set, studies are pooled, assessed and refined, thus the research embarked on the actual synthesis of data. To increase transparency, RAs and standards found as the result of this SRL is presented in
 370 table 1. For this purposes, the software Nvivo [57] has been used to code, label, and classify studies. Initially, all the keywords aforementioned has been created as nodes in the software, which are then associated to relevant sentences in studies. After coding all the studies, the findings have been synthesized to create theories, which in turn emerged themes and patterns. The findings gained from
 375 this SLR grounded the foundation for various aspect of the SLR development.

Study	Author	Year	Type
-------	--------	------	------

Towards a big Data reference architecture	[58]	2013	Master's Dissertation
A reference architecture for Big Data solutions introducing a model to perform predictive analytics using Big Data technology	[59]	2013	Conference Paper
IBM - Reference architecture for high performance analytics in healthcare and life science	[60]	2013	Book
Big data ecosystem reference architecture	[61]	2013	White Paper
A proposal for a reference architecture for long-term archiving, preservation, and retrieval of Big Data	[62]	2014	Conference Paper
Questioning the Lambda architecture; Kappa Architecture	[39]	2014	Blog
Defining architecture components of the Big Data Ecosystem	[63]	2014	Conference Paper
Oracle - Information Management and Big Data: A Reference Architecture	[64]	2014	White Paper
Big Data driven e-commerce architecture	[65]	2015	Journal Article
The solid architecture for real-time management of big semantic data	[66]	2015	Journal Article

Reference architecture and classification of technologies, products and services for big data systems	[67]	2015	Journal Article
A Reference Architecture for Big Data Systems	[68]	2016	Conference Paper
A reference architecture for Big Data systems in the national security domain	[69]	2016	Conference Paper
A Reference Architecture for Supporting Secure Big Data Analytics over Cloud-Enabled Relational Databases	[70]	2016	Conference Paper
SAP - NEC Reference Architecture for SAP HANA & Hadoop	[71]	2016	White Paper
Managing Cloud-Based Big Data Platforms: A Reference Architecture and Cost Perspective	[72]	2017	Journal Article
Scalable data store and analytic platform for real-time monitoring of data-intensive scientific infrastructure	[72]	2017	PhD Dissertation
A software reference architecture for semantic-aware Big Data systems; Bolster Architecture	[73]	2017	Journal Article
Simplifying big data analytics systems with a reference architecture	[54]	2017	Conference Paper
ISO/IEC/IEEE 42010:2011	[74]	2017	Standard

NIST Big Data Interoperability Framework: Volume 6, Big Data Reference Architecture	[75]	2018	White Paper
Towards a secure, distributed, and reliable cloud-based reference architecture for Big Data in smart	[18]	2019	Book Chapter
Reference Architectures and Standards for the Internet of Things and Big Data in Smart Manufacturing	[76]	2019	Conference Paper
Lambda architecture	[77]	2019	Conference Paper
ISO/IEC 20546:2019	[55]	2019	Standard
ISO/IEC TR 20547-1:2020	[56]	2020	Standard
NeoMycelia: A software reference architecture for big data systems	[3]	2021	Conference Paper
Smart transportation: A reference architecture for big data analytics	[53]	2021	Journal Article

Table 1: RAs and Standards found by the result of the SLR

3.4. Construction of the RA

Based on the themes, theories, and patterns realized in the previous steps, the process of RA construction took place. Integral to this step was the identification of elements that the RA should contain, how these elements should be synthesized, and how the RA can be portrayed and communicated. To describe our RA, we followed ISO/IEC/IEEE 42010 standard [74]. This standard

380

pivots on concrete architectures, so we did not 100% conform to it, but rather the good and relevant parts of it has been taken. For instance, architecture viewpoints, statement of corresponding rules, and expression of the architecture through architecture description languages (ADLs) have had direct aspects on
385 the construction of this RA.

A key challenge in the development of this RA was to strike a balance between the specificity of the micro patterns and approaches to system development and general architectural concepts that reflect a view of a the system as
390 an array of interrelated entities. Angelove et al [78] approached this problem by the means of interrogative through a defined framework that aims to guide the creation of RAs. Cloutier et al [17] suggest that a RA should entail technical, business and customer context views, whereas Vogel et al [79] provided classifies RA views based on the usage context, as industry specific, platform
395 specific, industry crosscutting and product line RAs.

Stricker et al [80] expressed their pattern-based RA by adhering several distinct views into one. Chang et al [75] presented NIST BD RA as system constituent of logical components connected though interoperability interfaces in several fabrics. On the other hand, ISO/IEC/IEEE 42010 refrains from using
400 phrases such as “technical architecture”, “physical architecture”, or “business architecture”.

Taking the best evidence from the available body of knowledge, We decided to adhere several views into one and it express the RA through a multi-layer modeling language called Archimate. Archimate is mature modeling language
405 developed by the Open Group that provides with a uniform representation of high-level architectural diagram aimed at portraying and delineating Enterprise architecture [81]. Archimate being listed as a standard architecture description language in ISO/IEC/IEEE 42010, is designed based on a set of related concepts that are specialized towards the system at different architectural layers. This
410 means that the architect is enhanced with an integrated architectural approach that visualizes and describes different architecture domains and their underlying relations [82]; [83].

Archimate utilizes service-orientation to distinguish and relate the application, business and technology layer and use realization relationships to create
415 relationship between concrete elements and more abstract elements across three layers. In addition, Archimate can be customized to account for varying needs of the architect.

3.5. Enabling RA with variability

Enabling RA with variability is an important process that helps with the
420 instantiation of it. This allows RA to remain useful as a priori artefact when it comes to country-specific regulations, and organizational compliance that constrain the architect design decisions [84].

Variability management has been studied in the domain of Business Process Management (BPM) [85]; [86]; [87] and Software Product Line Engineering (SPLE) [88][89]; [90]; [91]; [92]. In BPM, variability management revolves
425 around efficient handling of different variants in business processes, whereas in SPLE, variability management is about modifying and extending the software artefact to account of the requirements of a specific context.

Clear identification of variability and explicit communication of it improves
430 communication between stakeholders, allows for traceability between variation causes and effects and facilitates the decision making [93].

Variation points are decided based on the data collected in previous steps. Galster et al [24] suggest that there are three approaches to enabling variability;

1. Annotation of the RA
- 435 2. Variability views
3. Variability models

We could not find an in-detail explanation of how one should choose the appropriate variability enabling approach. Therefore, inspired by the works of Rurua et al [84], we decided to extend the RA with variability, by the means of
440 Archimate annotations. We have achieved this in two steps; first we developed a custom layer that represents focal variability concepts, and then we extended

the RA through annotation. The aim of this process is not find all variability points that may emerge in the usage context, but to provide with high-level system related architectural variabilities that an architect may consider for improvement of design and adoption of the RA.

The variability model is depicted in Fig 3 by the means of Archimate's motivation layer. This modeling is driven by the works of Pohl et al [88] and in specific their graphical notation of variability information, and Rurua et al [84] and in specific, their variability management concepts model.

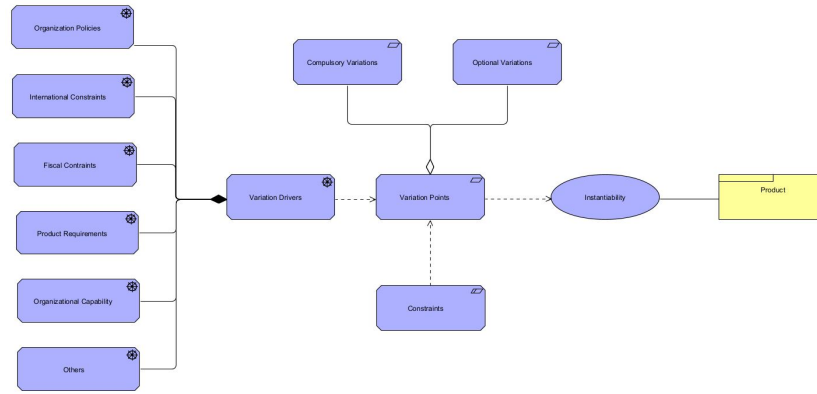


Figure 3: Variability management concepts model

3.6. Evaluation of the RA

Evaluation of the RA is to ensure that it has achieved the goals stated prior to development, to test its effectiveness and usability, and to make sure that it addresses the identified problems. Two fundamental pillars of the evaluation are the correctness and the utility of the RA and how efficient it can be adapted and instantiated [24]. The quality of RA can be assessed by how it can be transformed into an effective organization-specific concrete architecture. The fact that this RA is built upon former RAs helps making the evaluation steps easier as the research can get inspiration from other studies and their approach to evaluation [94].

460 Nevertheless, evaluation of the RAs is a well-known challenge among re-
searchers [34]; [95]; [96]; [58]. RAs and concrete architectures have distinct
qualities. They vary in at least 3 major ways;

1. RAs are of higher level of abstraction
2. in RAs stakeholders are not clearly grouped
- 465 3. RAs tend to be focused more on architectural qualities

While there are many well-established methods for assessing concrete archi-
tectures such as Scenario-based Architecture Analysis Method [97], Architecture
Level Modifiability Analysis [98], Performance Assessment of Software Architec-
ture [99], Architecture Trade-off Analysis Method [100], none of these methods
470 can be directly applied to evaluate RAs. To support this statement, three major
issues have been identified. These issues are as follows;

1. One of the main problems for applying existing evaluation methods to RA
is the lack of clearly defined group of stakeholders [30], while ATAM and
other methods are highly dependent on participation of stakeholders for
475 evaluation. Due to the level of abstractness of the RA, reaching various
group of stakeholders and persuade them to anticipate in the study, is
problematic and does not fit to the timeline of this study. Even more
notably, it is unlikely that all stakeholders will unite around a common
reference architecture as different members may or may not agree with the
480 overall idea of the RAs, may come from different backgrounds, and may
lack architectural visions
2. Evaluation frameworks and methods for concrete architectures make use
of scenarios. Howbeit due to RAs level of abstraction, creation of usable
scenario is difficult. Either a large set of scenarios should be developed
485 covering all the aspects of the RA with regards to specific domain, or a
more general scenarios should be developed to cover all the aspects. In the
first approach, a large number of scenarios, makes data analysis trouble-
some and a tedious process. Moreover, the order of prioritization of these

scenarios and defining them, and validating them is a problematic task.

490 In the second approach, due to the generality of the scenario, evaluation of effectiveness and usability of the RA becomes difficult and may become incomplete [95]. These challenges have been observed even in the evaluation of highly complex concrete architectures in information systems domain [101].

495 Based on the problems discussed above, available methods of architecture analysis are not sufficient in evaluating the RA. This has been addressed by various researchers in the industry.

In one study, Angelov et al [30], modified ATAM and extended it to resonate well with RAs. This process took place by invitation of representatives from leading industries for the evaluation process, and the selection of various contexts and defined scenarios for these contexts. Furthermore, ATAM has been extended to evaluate completeness, buildability and applicability. However, the selection of the right candidate and involving them in the process is a time-consuming and daunting task and may yield incomplete information. In addition, candidates maybe lacking architectural visions, increasing the threat to validity.

In addition to extending ATAM for RAs, Graaf et al [102] presented an evaluation approach in which SAAM is extended to help reduce the organizational impact of it. In Another study by Maier et al, Maier et al. (2013) as a post-graduate thesis in Eindhoven University of Technology, the evaluation of the RA has been conducted by mapping it against existing concrete architectures described in industrial whitepapers and reports. Along the lines, Galstar et al [24] suggested reference implementations, prototyping and incremental approach for the validation of the RA.

515 Rohling et al [103] have evaluated their RA by mapping it against the requirements set for the study. This was facilitated by the RA research methodology created by Nakagawa et al [104] and the complementary RAModel [26]. Inspired by all the studies listed, for the purposes of this study, we will first create a

prototype of the RA in an actual organizational setup and then we will use
520 ATAM to evaluate the concrete architecture.

4. Cybermycelium: A Domain Driven Distributed Reference Architecture for Big Data Systems

If our aspiration to enhance every business aspect with data needs to come to fruition, we need a different approach to data architecture. Traditional data
525 warehouse approaches to business intelligence, while have addressed the volume and computing aspect of data, they have failed to address other characteristics of it; heterogeneity and proliferation of data sources (variety), the speed at which data arrives and needs to be processed (velocity), the rate at which data mutates (variability), and the truth or quality of the data (veracity).

530 4.1. State of the art

The available body of knowledge and practice highlights 3 generations of big data architectures;

- 535 1. **Enterprise Data Warehouse:** this is perhaps one of the oldest approaches to business intelligence and data crunching and has existed even before the term big data was coined [105]. Usually developed as proprietary software, this data architecture pivots on enterprise data warehouse, ETL jobs, and a data visualization software such as Microsoft Power BI. As the data sources and consumers grow, this architecture suffers from hard to maintain ETL jobs, and visualizations that can be created and understood by a certain group of stakeholders, hindering data's potential
540 positive impact on business.
- 545 2. **Data Lake:** to address the challenges occurred in the first generation of data architectures, a new big data ecosystem emerged. This new ecosystem revolved around a data lake, in a way that there isn't as much transformations on the data, but rather everything is dumped into the data lake and retrieved when necessary. Although data lake architecture have

reached a higher level of success in comparison to the first generation of the data architectures, they are still far from optimal. As data consumer and data providers grow, data engineers will be immensely challenged to avoid creating a data swamp [106], and because there is usually no concept of data owner, the whole stack is usually operated by a group of hyper-specialized data engineers, creating silos, and barriers for gradual adoption. This also means various teams will often go into data engineers backlog and will not be in control of how and when they can consume the data they desire.

3. **Cloud Based Solutions:** Given the cost and complexity of running a data lake on-premise alongside the whole data engineering pipeline, and the substantial talent gap currently faced in the market [6], the third generation of big data architectures tend to revolve around as-a-service or on-demand cloud-based solutions. This generation of architecture tends to be leaning towards stream-processing with architectures such as Kappa or Lambda [107], or frameworks that unify batch and stream processing such as Apache Beam [108] or Databricks [109]. This is usually accompanied by cloud storage such as Amazon S3, and streaming technologies such as Amazon Kinesis. Whereas this generation tends to solve various issues regarding the complexity and cost of data handling and digestion, it still suffers from the same fundamental architectural challenges.

To discuss the integral facets that embroil these architectures, one must look at the characteristics of these architectures and the ways in which they achieve their ends, with quality attributes surrounding it. Except for one case [3], all the architectures and RAs found as the result of this study, were designed underlying monolithic data pipeline architecture with four major components being data consumer, data processing, data infrastructure and data providers.

The process of turning data into actionable insights in these architectures usually follow a similar lifecycle;

1. **Data Ingestion:** system beings to ingest data from all corners of the

enterprise, including both transactional, operational and external data. For instance, in a practice management software for veterinaries, data platform can ingest and persist transactional data such as 'user interaction with therapeutics', 'number of animals diagnosed', or 'number of invoices created' and 'medicines dispensed'.

2. **Data Transformation:** data captured from the previous step is then cleansed for duplication, quality, and potentially scrubbed for privacy policies. This data then goes through a multifaceted enrichment process to facilitate data analysis. For instance, a journey of the veterinary nurse can be captured at every stage, enriched with demographics and animal breed for regression analysis and aggregate views.
3. **Data Serving:** at this stage, data is ready to be served to diverse array of needs ranging from machine learning to marketing analytics, to business intelligence to product analysis and customer journey optimization. In the 'veterinary practice management software' the data platform can provide real-time data through event backbone system such as Kafka about customers who have applied and have been dispensed Restricted Veterinary Medicine (RVM) to make sure that these transactions comply with the conditions of the registration of these products.

The lifecycle depicted is indeed a high-level abstract view of prevalent big data systems. Howbeit, it highlights an important matter; these systems are all operating underlying monolithic data pipeline architecture that tends to account for all sorts of data. This means, data that logically belong to different domains are now all lumped together and crunched in one architectural constructs, making maintainability and scalability a daunting task [110].

While architectures in software engineering have gone through series of evolution in the industry, adopting a more decentralized and distributed approaches such as microservice architecture, event driven architectures, reactive systems, and domain driven design [111], the data engineering, and in specific big data ecosystems do not seem to be adopting many of these patterns. Evidence col-

lected from this study have proven that attention to decentralized big data systems, metadata, and privacy is deficient. Therefore, the whole idea of 'monolithic data pipeline architecture with no clearly defined domains and ownership' brings significant challenges to design, implementation, maintenance and scaling of big data systems.

Suppose company A would want to adopt a big data initiative to embark on this complex endeavour, what would be the first step ? does the company have to worry about high-throughput stream processing ? does it have to worry about regional privacy regulations? does it have to worry about cost-efficient batch processing? perhaps yes to all of these questions, but what's even more integral to the success of the whole endeavour is the underlying architecture that governs the entire system, its components, their relations to each other, data flow and principles and standards that govern the quality attributes and evolution of the system.

This architecture and design process if done underlying current prevalent approaches, can bring about colossal lost, and leave many managers disappointed. In fact, we don't claim that all these architectures will fail, perhaps some have proven to be successful in a specific context. There are two threats to maintainability and scalability of these systems;

1. **Data source proliferation:** as the big data system grows and more data sources are added, the ability to ingest, process, and harmonize all these data in one place diminishes. This in turn, reduces maintainability, makes company reliant on lead data engineers who built the infrastructure, and makes scaling these systems very difficult. The proliferation of data sources if not managed carefully, can result in data swamps as well, which makes understanding data domains, and providing data as a service a complicated task.
2. **Data consumer proliferation:** organizations that utilize rapid experimentation approaches such Hypothesis-Driven Development and Continuous Delivery [112] constantly introduce new use cases for data to be con-

sumed in different domains. This means that variability of the data rises, and the sum of aggregations, projections, and slices increases, which in turn adds more work to the backlog of the data engineering team, slowing
640 down the process of serving the data to consumers. Inability to account for the data consumer demands can be a point of friction in organizations [110].

To address these challenges, the lead architect or the architecture governance group will then have to choose the right architectural quanta to segregate the
645 monolith. According to Ford et al [113], an architectural quantum is a component of a system with high functional cohesion that is independently deployable, this is also referred to as a service [114]. The main motivation for segregating the monolith into its architectural quantum, is to parallelize work in various business domain, to reach a better velocity, reduce cost, promote ownership, increase performance and reach higher operational scalability.
650

Currently, these architectures are usually segregated into pipelines that each process data differently. While each pipeline has its own responsibility to handle various aspect of the big data system, there is still a high coupling between the pipelines, as 'data cleansing' phase cannot start after 'data ingestion'. This
655 coupling is even more highlighted when the company is at the stage of rapid experimentation with data sources and would like to explore new domains of insight generation, and this in turn means that delivering new features and values is orthogonal to the axis of change.

Using the same practice management software example, given that a new
660 class of animals (equine animals as opposed to small animals) should be incorporated for data analysis; the data engineering team should then modify and extend the whole pipeline of ingest, process and serve to account for the particularity of the data captured regarding this new class of animals. New ingestion services required, the schema might change, the veracity checking mechanisms
665 might differ, cleansing varies and more. This implies an end-to-end dependency which affects external teams, slows down processes and make maintenance grad-

ually harder. This implies that using pipeline as an architectural quantum in such a coupled way is perhaps not the most efficient architecture to big data systems.

670 Another major issue with the current architectural approaches is that data engineering is usually confined into a team of hyper-specialized individuals who are siloed from the operational units of the organization. These teams, being fully responsible for creating the infrastructure for data processing, are often absent of business knowledge and the domain, which limits their productivity. 675 These individual usually have a limited understanding of the data sources, data provenance, data consumers, the changing nature of the business domains, the overall product vision, and the application side of things, yet they are responsible to provide data for a large array of analytical and operational needs in a timely manner. For instance, given a context in which a product owner and application 680 developer can cooperate, the synergy between the data engineer, application developer, and product owner can bring about more mature decisions that can align the requirements across various technical and logical domains and allow for various stakeholders to contend and communicate their concerns.

Whereas there could be other factors to be discussed deeply in this paper 685 regarding current big data architectures, our aim is not to explore any further and emphasize more on the solution artefact we've designed to address some of these challenges.

5. A paradigm shift: distributed domain driven architecture

Based on the premises discussed priori to this section, one can infer that the 690 idea of monolithic and centralized data pipelines that are highly coupled and operated by silos of hyper-specialized big data engineers has limitations and can bring organizations into a bottleneck .

We therefore, explore a domain driven distributed and decentralized architecture for big data systems and posit that this architecture can address some of 695 the challenges discussed. This idea is inspired by the advancements in software

engineering architecture, and in specific event-driven microservice architecture [115], domain driven design [116], and reactive systems [117].

Data usually comes into two different flavours; 1) operational data which serves the need of an application, facilitates logic, and can include transactional data and 2) analytical data which usually has the temporality to it, and is aggregated to provide with insights.

These two different flavours, despite being related, have different characteristics and trying to lump them together may result in a morass. To this end, Cybermycelium realizes the varying nature between these two planes and respects the difference. Cybermycelium aims to transfigure current architectural approaches by proposing an inversion of control, and a topology based on product domains and not the technology [118]. Our proposition is that handling two different archetypes of data, should not necessarily result in siloed teams, heavy backlogs, and a coupled implementation.

To further elucidate on this matter, we take the example of the microservice architecture. As the industry sailed away from the monolithic n-tier architectures into Service Oriented Architectures (SOA), organizations faced a lot of challenges. One prevalent issue was around the maintenances of Enterprise Service Bus (ESB) or SOA bus, which is the locus of aggregation. While the aggregation layer could be written very thin, the reality is that the transformation of XML and logical operations started to bloat the SOA bus. This added a new level of coupling between internal and external elements of the system as a whole [119]; [120]; [121].

Microservices architecture, being the evolution of SOA, move away from smart pipelines into dumb pipelines and smart services removing the need for the locus of aggregation and control. Moreover, there was no business logic written in the pipelines, and each service was segregated usually with the help of domain driven design.

Whereas microservices architecture still have its challenges, the gradations of software architectures in software engineering industry can be analogous to the data engineering domain. One can perceive the pipeline architecture and its

coupling nature similar to SOA and the business logic written to connect the services in the SOA bus.

Based on these premises we posit 4 underpinning principles for Cybermycelium;

- 730 1. Distributed Domain driven services with bounded context
2. Data as a service
3. Data infrastructure automation
4. Governance through a federation service

5.1. Distributed Domain driven services with bounded context

735 Integral to Cybermycelium, is the distribution and decentralization of services into domains that have clear bounded context. Perhaps one the most challenging things one might face when it comes to architecting a distributed system is; based on what architectural quanta should we break down the system. This issue has been repeatedly discussed for example among adopters of microservices architecture . Cybermycelium, inspired by the concept of domain-
740 drive design, tends to sit data close to the product domain that relates to it. This implies that data inheres in the product domain and as a facet of it [46].

This is mainly driven by the fact that most organizations today are decomposed based on their products. These products are the capability of the business
745 that are segregated into various domains. Domain's bounded context is operated by various teams with different visions and concerns, incorporating data into a bounded context can result in a synergy that can improve the management of evolution and continuous change. This can be micro, such as application developers communicating with data engineers about collecting user data in a
750 nested data structures or in flat ones, or macro, such as application developers thinking about redesigning their graphql schema in an intermediary layer that may affect the data engineers ingestion services.

5.2. Data as a service:

Data can be conceived as the fourth dimension of a product next to UI/UX,
755 business and application (5.2). Each domain provides its data as a service. This

data is consisting of both operational and analytical data. This also implies that any friction and coupling between data is removed. For instance, the 'invoice' domain will provide the transactional data about number of invoices and total of discounts with analytical data such as which practices have created what
760 number of invoices in what period of time.

Product Domain

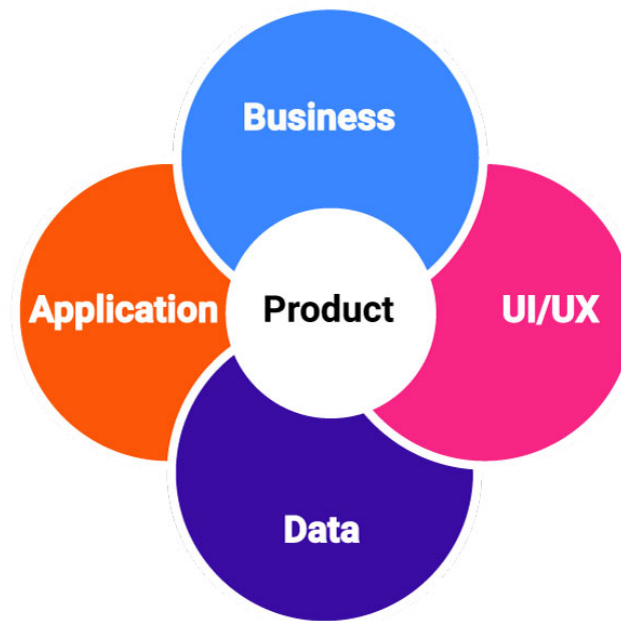


Figure 4: Product Facets

However this data-as-a-service model should be carefully implemented to account for explorability, discoverability, security, and quality. The data provided as a service should have the identical qualities to customer-facing products. This also implies, that a product owner should now treat data facet as an aspect of
765 the product and employ objective measures that assure the desired quality. These measure can be net promoter scores from data consumers, data provenance, decreased lead time. Product owners, in addition to the application and design aspect of the product, must now incorporate this new facet, and try to

understand the needs of the data consumers, how they consume the data, and
770 what are the common tools and technologies they utilized to consume the data.
This knowledge can help shaping better interfaces for the product.

Product domains may also need to ingest data from upstream domains, and
this requires the definition of clear interfaces. Furthermore, each domain should
also account for metadata. Metadata is derived from the nature of the product
775 and its data lifecycle. Data can be ingested and served in various forms such
as tables, graphs, JSON, events, and many more; but in order for the data
to be useful for analytical purposes, there is a need to associate data with its
corresponding metadata that encompasses semantics, and history.

5.3. *Data infrastructure automation*

780 As the number of product domain increases, the effort required to build,
deploy, execute, and monitor services increases. This includes the data pipelines
required for that product domain to carry out its functions. The platform skills
required for these kinds of work is usually found in Devops engineers and site
reliability engineers. Application developers and data engineers are usually not
785 adept at carrying out such workloads in an efficient manner. For this reason,
there is a need for highly abstract reusable infrastructural components that can
be easily utilized. This implies that teams should be equipped with required
infrastructure as a service, that can be easily employed to account for big data
needs.

790 One way to provision such infrastructure as a service, is to utilize infras-
tructure as a code software tools like Terraform [122]. Besides, data infrastruc-
ture may be extended based on currently running infrastructure for application
payloads. However, this might be challenging, as the big data ecosystem is
growing rapidly, and while a software application might be running in an EC2
795 worker node in an EKS cluster on Amazon, the big data system maybe running
Databricks, a CDP solution like Segment [123], HDFS [124], or Amazon Kine-
sis. This brings the challenge of composing data and application infrastructure
together to provide with coherent, cost efficient interoperable infrastructure.

Nevertheless, this should not be a daunting task, as one can simply extend
800 the worker nodes map in the EKS code written for Terraform to add a new
node in the network, which installs Databricks through a Helm Chart [125]. In
addition, the data infrastructure should be accompanied with proper tooling.
Tools like GNU Make [126] makes it quite easy for developers and data engi-
neers to deploy infrastructure as they demand. A mature infrastructure as a
805 service should provide the team with core infrastructures such as big data stor-
age, stream processing services, batch processing services, event backbones or
message queues, and data integration technologies.

5.4. Governance through a federation service

The last principle of Cybermycelium is the global governance or the global
810 standardization of the services. This principle is perhaps a lesson learnt from
the studied application of Miroservices architecture in the industry [111]. Dis-
tributed architectures are made up of independent collection of nodes, with
distinct lifecycle that are deployed separately and are owned by various teams.
As the number of these services grow, and the interconnections increase, the
815 challenge of maintaining and scaling the system increases. This means services
need to interoperate, provide services, ingest data from other services, perform
graph or set operations in a timely manner and do stream processing.

In order to scale and maintain these independency deployed yet intercon-
nected services, Cybermycelium needs a governance model that embrace domain
820 autonomy, distribution and decentralization, automation and Devops, and in-
creased interoperability through federated government. This requires a shift
in thinking, which obsoletes many prevalent assumptions of software and data
engineering. The point of federation is not to suppress or kill the creativity and
innovation of the teams, but rather, introduction of global contracts and stan-
825 dards that are in-line with company's resources and vision. This therefore can
be a challenging task for software architects to find the equilibrium bet ween
right amount of centralization and decentralization. For instance, semantic re-
lated metadata can be left to the product domain to decide, whereas policies and

standards for metadata collection should be global. This is somewhat analogous
 830 to architectural principles in TOGAF's ADM [127].

The definition of these standards is up to the architecture, or architectural governance group, and is usually achieved through service level objectives (SLOs) or well-defined contracts and standards.

5.5. The Artefact:

835 After having discussed many kernel and design theories, the necessary theoretical foundation is created for the design and development of the artefact. Cybermycelium is created with Archimate and displays the RA in various layers. For the sake of completion, and as every software is designed to account for a business need, we have assumed a very simple big data business process.
 840 While this business layer could vary in different context, Cybermycelium should be able to have the elasticity required to account for various business models.

Cybermycelium is made up of 11 main components and 7 variable components as depicted in figure 5.5.

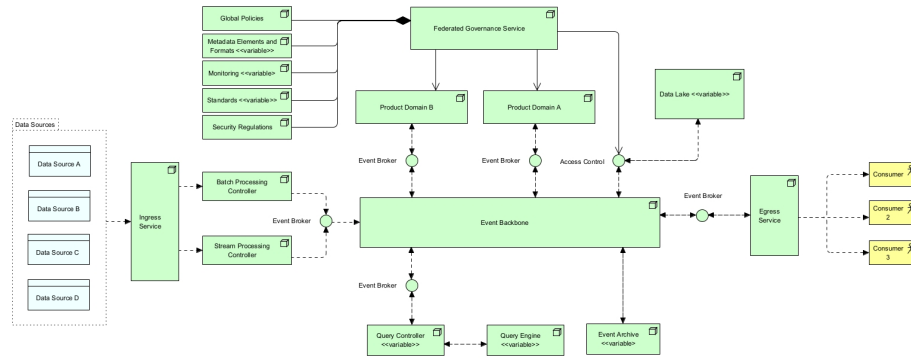


Figure 5: Cybermycelium Big Data Reference Architecture

For the purposes of effective modeling and abstraction, we've sub-diagrammed
 845 the product domain, which is portrayed in figure 5.5

The main elements are;

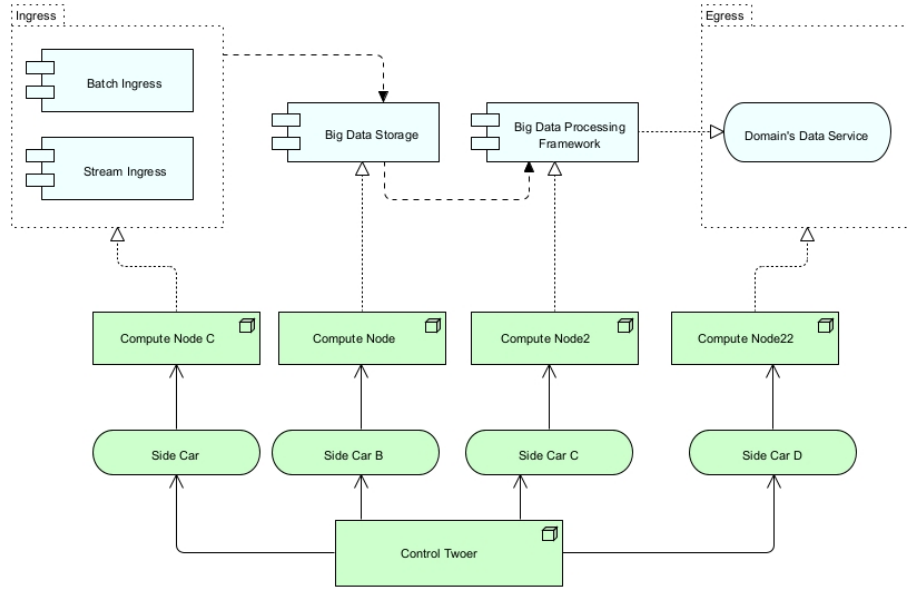


Figure 6: Cybermycelium Product Domain Design

1. **Ingress Service:** The ingress service is responsible for exposing the necessary port and endpoint for the data to flow to the system. Depending on the nature of the request, the ingress service will load balance to either batch processing controller or stream processing controller. It is essential for the ingress service to operate asynchronously to avoid any potential choke points. In addition, ingress handles the SSL termination, and potentially name-based virtual hosting. Ingress has several benefits. Firstly, it helps with security by consolidating by preventing port proliferation and direct access to services. Secondly, it help with performance by distributing requests based on their nature, and SSL termination.
2. **Batch Processing Controller:** Batch processing controller is responsible for dispatching batch events to the event backbone. This service should be a small service (could be a Lambda) with the main responsibility of receiving a request for batch processing and dispatching an event to the event broker. Because the nature of the request is of type batch and has been clearly distinguished by the ingress, batch processing controller can

dispatch events in bulk and asynchronously. This is the main difference of this service to stream processing controller. Batch processing controller can execute other non compute-intensive tasks such as scrubbing properties from the given data or adding headers.

3. **Stream Processing Controller:** Stream processing controller is responsible for dispatching streaming events to the event backbone through the event broker. This service has been segregated from the batch service as it has to account for a different nature of events. Streams are synchronous in nature, and can require high-throughput. This service is a small service as well, but non-heavy computations such as enabling stream provenance, and one-pass algorithms can be utilized.
4. **Event Broker:** Event brokers are designed to achieve 'inversion of control'. As the company evolves and requirements emerge, the number of nodes in the service increases, new regions of operations may be added, and new events might need to be dispatched. As each service has to communicate with the rest through the event backbone, each service will be required to implement it's own event handling module. This can easily turn into a spaghetti of various implementation by various teams, and can even cause bugs and unexpected behaviors. To overcome this challenge, an event broker is introduced to each main service of the architecture, plus the ingress and egress. Each service connects to its local event broker and publishes and subscribes to events through that broker. One of the key success criteria of the event broker is a unified interface that sits at a right level of abstraction to account for all services of the architecture. Event brokers, being environmentally agnostic can be deployed to any on-premise, private or public infrastructure. Event brokers can also account for more dynamism by learning which events should be routed to which consumer applications. Even brokers do also implement circuit breaking, which means if the service they have to broke to is not available and does not respond for a certain amount of time, the broker establishes unavailability as a service to the system, so no further requests come through.

This is essential to preventing a ripple effect over the whole system if one system fails.

895
5. **Event Backbone:** This is the heart of the Cybermycelium, facilitating communication among all the nodes. Event backbone in itself should be distributed and ideally clustered to account for the ever-increasing scale of the system. Communication occurs as choreographed events from services analogous to a dance troupe. In a dance troupe, the members respond to the rhythm of the music by moving according to their specific roles. In here, each service (dancer) listens and reacts to the event backbone (music) and takes the required action. This means services are only responsible for dispatching events in a 'dispatch and forget' model, and subscribe to the topics that are necessary to achieve their ends. Event backbone thus ensures a continues flow of data among services so that all systems are in the correct state at all times.

900
905
6. **Egress Service:** The egress service is responsible for providing necessary APIs for the consumers of the system to request data in demand. This is a self-serve data model in which data scientists or business analyst can readily request data from various domains based on the data catalogue. Clients can first request for a data catalogue and then use the catalogue to request for the product domain that accounts for the desired data. This request can include several data products. Egress is responsible to route the request to the data catalogue, and to corresponding product service mesh' in order to resolve values. The egress realizes the address to service meshes and other services through the data catalog. The egress service should cache the resolved addresses and values in order to increase performance and response time. An architect can even choose the implement a complete query cache component inside the egress service, however that will make maintenance potentially more difficult and may result in bottlenecks.

910
915
920
7. **Product Domain Service Mesh:** As previously discussed a product is a capability of the business, and each product has its own domain consisting

925 of the bounded context and the ubiquitous language. In system and ar-
chitectural terms, these domains are implemented as a service mesh. Each
service mesh is made up of a batch ingress, stream ingress, big data stor-
age, big data processing framework, domain’s data service, the required
compute nodes to run these services, a side car per service and a control
930 tower. These components provide the necessary means for the domain to
achieve its ends.

8. **Federated Governance Service:** Evidently, Cybermycelium is a dis-
tributed architecture that encompasses variety of independent services
with independent lifecycle that are built and deployed by independent
935 teams. Whereas teams have their autonomy established, in order to
avoid haphazard, out-of-control and conflicting relations, there should be
a global federated governance that aims to standardize these services.
This will facilitate the interoperability between services, communication,
aggregates, and even allows for a smoother exchange of members across
940 teams. This also means the most experienced people at a company such
as technical leads and lead architects will prevent potential pitfalls that
more novice engineers may fall into. However the aim of this service is not
centralize control in anyway, as that would be going a step backward into
the data warehouse era. The aim of this service is to allow autonomous
945 flow in the river of standards and policies that tend to protect company
from external harm. For instance, failing to comply to GDPR while oper-
ating in europe can sets forth fines up to 10 million euros, and this may
not be something that novice data engineers or application developers are
fully aware of.

950 Service mesh is a suitable architectural construct for product domains due
to several reasons which are discussed later in this paper. Service mesh,
by utilizing two integral components which are the control tower and the
side cars, abstract away all the platform work that data engineers or appli-
cation developers are not very skilled in. This implies that engineers can
955 concentrate at what they excel at, without having to deal with platform

noise. Sidecars

9. **Federated Governance Service:**

10. **Global Policies:**

11. **Security Regulations:**

960 References

- [1] P. Ataei, A. T. Litchfield, Big data reference architectures, a systematic literature review.
- [2] B. B. Rad, P. Ataei, The big data ecosystem and its environs, International Journal of Computer Science and Network Security (IJCSNS) 17 (3) 965 (2017) 38.
- [3] P. Ataei, A. Litchfield, Neomycelia: A software reference architecture for big data systems, in: 2021 28th Asia-Pacific Software Engineering Conference (APSEC), IEEE Computer Society, Los Alamitos, CA, USA, 2021, pp. 452–462. doi:10.1109/APSEC53868.2021.00052.
970 URL <https://doi.ieeecomputersociety.org/10.1109/APSEC53868.2021.00052>
- [4] I. L. Stats, Internet live stats (2019).
URL <https://www.internetlivestats.com/>
- [5] M. Lycett, ‘datafication’: Making sense of (big) data in a complex world 975 (2013).
- [6] B. B. Rada, P. Ataeib, Y. Khakbizc, N. Akbarzadehd, The hype of emerging technologies: Big data as a service.
- [7] M. Huberty, Awaiting the second big data revolution: from digital noise to value creation, Journal of Industry, Competition and Trade 15 (1) (2015) 980 35–47.

- [8] M. technology review insights in partnership with Databricks, Building a high-performance data organization (2021).
URL <https://databricks.com/p/whitepaper/mit-technology-review-insights-report>
- 985 [9] N. Partners, Big data and ai executive survey 2021 (2021).
URL https://www.supplychain247.com/paper/big_data_and_ai_executive_survey_2021/pragmadik
- [10] M. Analytics, The age of analytics: competing in a data-driven world, Tech. rep., Technical report, San Francisco: McKinsey & Company (2016).
- 990 [11] H. Nash, Cio survey 2015, Association with KPMG.
- [12] N. Singh, K.-H. Lai, M. Vejvar, T. Cheng, Big data technology: Challenges, prospects and realities, IEEE Engineering Management Review.
- [13] B. B. Rad, P. Ataei, Evaluating major issues regarding reliability management for cloud-based applications, IJCSNS 17 (7) (2017) 168.
- 995 [14] I. Gorton, J. Klein, Distribution, data, deployment, STC 2015 (2015) 78.
- [15] S. Nadal, V. Herrero, O. Romero, A. Abelló, X. Franch, S. Vansummeren, D. Valerio, A software reference architecture for semantic-aware big data systems, Information and software technology 90 (2017) 75–92.
- [16] R. K. Len Bass, Dr. Paul Clements, Software Architecture in Practice (SEI Series in Software Engineering) 4th Edition, Addison-Wesley Professional;
1000 4th edition, 2021.
- [17] R. Cloutier, G. Muller, D. Verma, R. Nilchiani, E. Hole, M. Bone, The concept of reference architectures, Systems Engineering 13 (1) (2010) 14–27.
- 1005 [18] J. Kohler, T. Specht, Towards a secure, distributed, and reliable cloud-based reference architecture for big data in smart cities, in: Big Data Analytics for Smart and Connected Cities, IGI Global, 2019, pp. 38–70.

- [19] M. Derras, L. Deruelle, J.-M. Douin, N. Levy, F. Losavio, Y. Pollet, V. Reiner, Reference architecture design: A practical approach, in: IC-SOFT, pp. 633–640.
1010
- [20] J. Bayer, T. Forster, D. Ganesan, J.-F. Girard, I. John, J. Knodel, R. Kolb, D. Muthig, Definition of reference architectures based on existing systems, Fraunhofer IESE, March.
- [21] V. Stricker, K. Lauenroth, P. Corte, F. Gittler, S. De Panfilis, K. Pohl, Creating a reference architecture for service-based systems—a pattern-based approach, in: Towards the Future Internet, IOS Press, 2010, pp. 149–160.
1015
- [22] E. Gamma, R. Helm, R. Johnson, R. E. Johnson, J. Vlissides, et al., Design patterns: elements of reusable object-oriented software, Pearson Deutschland GmbH, 1995.
1020
- [23] E. Y. Nakagawa, R. M. Martins, K. R. Felizardo, J. C. Maldonado, Towards a process to design aspect-oriented reference architectures, in: XXXV Latin American Informatics Conference (CLEI) 2009, 2009.
- [24] M. Galster, P. Avgeriou, Empirically-grounded reference architectures: a proposal, in: Proceedings of the joint ACM SIGSOFT conference—QoSA and ACM SIGSOFT symposium—ISARCS on Quality of software architectures—QoSA and architecting critical systems—ISARCS, 2011, pp. 153–158.
1025
- [25] E. Y. Nakagawa, M. Guessi, J. C. Maldonado, D. Feitosa, F. Oquendo, Consolidating a process for the design, representation, and evaluation of reference architectures, in: 2014 IEEE/IFIP Conference on Software Architecture, IEEE, 2014, pp. 143–152.
1030
- [26] E. Y. Nakagawa, F. Oquendo, M. Becker, Ramodel: A reference model for reference architectures, in: 2012 Joint Working IEEE/IFIP Conference on

- 1035 Software Architecture and European Conference on Software Architecture,
IEEE, 2012, pp. 297–301.
- [27] J. F. M. Santos, M. Guessi, M. Galster, D. Feitosa, E. Y. Nakagawa, A
checklist for evaluation of reference architectures of embedded systems
(s)., in: SEKE, Vol. 13, 2013, pp. 1–4.
- 1040 [28] M. Derras, L. Deruelle, J. M. Douin, N. Levy, F. Losavio, Y. Pol-
let, V. Reiner, Reference architecture design: a practical approach,
in: 13th International Conference on Software Technologies (ICSOFT),
SciTePress-Science and Technology Publications, 2018, pp. 633–640.
- [29] I. WG, Iso/iec 26550: 2015-software and systems engineering-reference
1045 model for product line engineering and management, ISO/IEC, Tech. Rep.
- [30] S. Angelov, J. J. Trienekens, P. Grefen, Towards a method for the eval-
uation of reference architectures: Experiences from a case, in: European
Conference on Software Architecture, Springer, 2008, pp. 225–240.
- [31] S. Angelov, J. J. Trienekens, P. Grefen, Extending and adapting the archi-
1050 tecture tradeoff analysis method for the evaluation of software reference
architectures.
- [32] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere,
The architecture tradeoff analysis method, in: Proceedings. fourth ieee
international conference on engineering of complex computer systems (cat.
1055 no. 98ex193), IEEE, 1998, pp. 68–78.
- [33] S. Angelov, P. Grefen, D. Greefhorst, A classification of software reference
architectures: Analyzing their success and effectiveness, in: 2009 Joint
Working IEEE/IFIP Conference on Software Architecture & European
Conference on Software Architecture, IEEE, 2009, pp. 141–150.
- 1060 [34] S. Angelov, P. Grefen, An e-contracting reference architecture, Journal of
Systems and Software 81 (11) (2008) 1816–1844.

- [35] D. Hollingsworth, U. Hampshire, Workflow management coalition: The workflow reference model, Document Number TC00-1003 19 (16) (1995) 224.
- 1065 [36] H. Zimmermann, Osi reference model-the iso model of architecture for open systems interconnection, IEEE Transactions on communications 28 (4) (1980) 425–432.
- [37] OATH, Oath reference architecture, release 2.0 initiative for open authentication, OATH.
- 1070 URL <https://openauthentication.org/wp-content/uploads/2015/09/ReferenceArchitectureVersion2.pdf>
- [38] A. L. Pope, The CORBA reference guide: understanding the common object request broker architecture, Addison-Wesley Longman Publishing Co., Inc., 1998.
- 1075 [39] J. Kreps, Questioning the lambda architecture, Online article, July 2015. URL <https://www.oreilly.com/radar/questioning-the-lambda-architecture/>
- [40] M. Press, L. Joyner, G. Malcolm, Application Architecture for .NET: Designing Applications and Services, Microsoft Press, 2002.
- 1080 [41] D. Greefhorst, P. Gehner, Achmea streamlines application development and integration, Via Nova Architectura.
- [42] D. Greefhorst, Een applicatie-architectuur voor het web bij de bank—de pro’s en contra’s van toestandsloosheid, Software Release Magazine 2.
- [43] H. Wu, A reference architecture for Adaptive Hypermedia Applications, Citeseer, 2002.
- 1085 [44] A. H. Norta, Exploring dynamic inter-organizational business process collaboration (2007).

- [45] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, et al., An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems, in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, 2019, pp. 3–18.
- [46] R. Laigner, Y. Zhou, M. A. V. Salles, Y. Liu, M. Kalinowski, Data management in microservices: State of the practice, challenges, and research directions, arXiv preprint arXiv:2103.00170.
- [47] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina, Microservices: yesterday, today, and tomorrow, Present and ulterior software engineering (2017) 195–216.
- [48] Qcon software conferences (2022).
URL <https://qconferences.com/>
- [49] State of data mesh 2022 (2022).
URL <https://www.thoughtworks.com/about-us/events/state-of-data-mesh-2022>
- [50] Worldwide software architecture summit’21 (2021).
URL https://events.geekle.us/software_architecture/
- [51] Kafka summit europe 2021 (2021).
URL <https://www.confluent.io/events/kafka-summit-europe-2021/>
- [52] D. Moher, L. Shamseer, M. Clarke, D. Ghersi, A. Liberati, M. Petticrew, P. Shekelle, L. A. Stewart, Preferred reporting items for systematic review and meta-analysis protocols (prisma-p) 2015 statement, Systematic reviews 4 (1) (2015) 1–9.

- [53] C. Castellanos, B. Perez, D. Correal, Smart transportation: A reference architecture for big data analytics, in: Smart Cities: A Data Analytics Perspective, Springer, 2021, pp. 161–179.
- [54] G. M. Sang, L. Xu, P. d. Vrieze, Simplifying big data analytics systems with a reference architecture, in: Working Conference on Virtual Enterprises, Springer, 2017, pp. 242–249.
- [55] I. O. for Standardization (ISO/IEC), Iso/iec 20546:2019 (2019).
URL <https://www.iso.org/standard/68305.html>
- [56] I. O. for Standardization (ISO/IEC), Iso/iec tr 20547-1:2020 (2020).
URL <https://www.iso.org/standard/71275.html>
- [57] Unlock insights in your data with the best qualitative data analysis software (2022).
URL <https://www.qsrinternational.com/nvivo-qualitative-data-analysis-software/home>
- [58] M. Maier, A. Serebrenik, I. Vanderfeesten, Towards a big data reference architecture, University of Eindhoven.
- [59] B. Geerdink, A reference architecture for big data solutions introducing a model to perform predictive analytics using big data technology, in: 8th international conference for internet technology and secured transactions (ICITST-2013), IEEE, 2013, pp. 71–76.
- [60] D. Quintero, F. N. Lee, et al., IBM reference architecture for high performance data and AI in healthcare and life sciences, IBM Redbooks, 2019.
- [61] B. Levin, Big data ecosystem reference architecture, Microsoft Corporation.
- [62] P. Viana, L. Sato, A proposal for a reference architecture for long-term archiving, preservation, and retrieval of big data, in: 2014 IEEE 13th

- 1140 International Conference on Trust, Security and Privacy in Computing
and Communications, IEEE, 2014, pp. 622–629.
- [63] Y. Demchenko, C. De Laat, P. Membrey, Defining architecture components of the big data ecosystem, in: 2014 International conference on collaboration technologies and systems (CTS), IEEE, 2014, pp. 104–112.
- 1145 [64] D. Cackett, Information management and big data, a reference architecture, Oracle: Redwood City, CA, USA.
URL <https://www.oracle.com/technetwork/topics/entarch/articles/info-mgmt-big-data-ref-arch-1902853.pdf>
- [65] A. Ghandour, Big data driven e-commerce architecture, International
1150 Journal of Economics, Commerce and Management 3 (5) (2015) 940–947.
- [66] M. A. Martínez-Prieto, C. E. Cuesta, M. Arias, J. D. Fernández, The solid architecture for real-time management of big semantic data, Future Generation Computer Systems 47 (2015) 62–79.
- [67] P. Pääkkönen, D. Pakkala, Reference architecture and classification of
1155 technologies, products and services for big data systems, Big data research 2 (4) (2015) 166–186.
- [68] G. M. Sang, L. Xu, P. De Vrieze, A reference architecture for big data systems, in: 2016 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA), IEEE, 2016, pp. 370–
1160 375.
- [69] J. Klein, R. Buglak, D. Blockow, T. Wuttke, B. Cooper, A reference architecture for big data systems in the national security domain, in: 2016 IEEE/ACM 2nd International Workshop on Big Data Software Engineering (BIGDSE), IEEE, 2016, pp. 51–57.
- 1165 [70] A. Cuzzocrea, A reference architecture for supporting secure big data analytics over cloud-enabled relational databases, in: 2016 IEEE 40th

Annual Computer Software and Applications Conference (COMPSAC),
Vol. 2, IEEE, 2016, pp. 356–358.

- [71] Sap - nec reference architecture for sap hana & hadoop.

1170 URL [https://www.scribd.com/document/418835912/](https://www.scribd.com/document/418835912/Whitepaper-NEC-SAPHANA-Hadoop)
Whitepaper-NEC-SAPHANA-Hadoop

- [72] L. Heilig, S. Voß, Managing cloud-based big data platforms: a reference
architecture and cost perspective, in: Big data management, Springer,
2017, pp. 29–45.

- 1175 [73] S. Nadal, V. Herrero, O. Romero, A. Abelló, X. Franch, S. Vansummeren,
D. Valerio, A software reference architecture for semantic-aware big data
systems, Information and software technology 90 (2017) 75–92.

- [74] I. International Organization for Standardization (ISO/IEC), Iso/iec/ieee
42010:2011 (2017).

1180 URL <https://www.iso.org/standard/50508.html>

- [75] W. L. Chang, D. Boyd, Nist big data interoperability framework: Volume
6, big data reference architecture, Report (2018).

- [76] P. Ünal, Reference architectures and standards for the internet of things
and big data in smart manufacturing, in: 2019 7th International Confer-
ence on Future Internet of Things and Cloud (FiCloud), IEEE, 2019, pp.
1185 243–250.

- [77] M. Kiran, P. Murphy, I. Monga, J. Dugan, S. S. Baveja, Lambda archi-
tecture for cost-effective batch and speed big data processing, in: 2015
IEEE International Conference on Big Data (Big Data), IEEE, 2015, pp.
1190 2785–2792.

- [78] S. Angelov, P. Grefen, D. Greefhorst, A framework for analysis and design
of software reference architectures, Information and Software Technology
54 (4) (2012) 417–431.

- 1195 [79] O. Vogel, I. Arnold, A. Chughtai, E. Ihler, T. Kehrer, U. Mehlig, U. Zdun, Software-architektur: Grundlagen–konzepte, Praxis 2.
- [80] V. Stricker, K. Lauenroth, P. Corte, F. Gittler, S. De Panfilis, K. Pohl, Creating a reference architecture for service-based systems-a pattern-based approach, in: Future Internet Assembly, pp. 149–160.
- 1200 [81] M. Lankhorst, A language for enterprise modelling, in: Enterprise Architecture at Work, Springer, 2013, pp. 75–114.
- [82] M. M. Lankhorst, H. A. Proper, H. Jonkers, The anatomy of the archimate language, International Journal of Information System Modeling and Design (IJISMD) 1 (1) (2010) 1–32.
- 1205 [83] W. Engelsman, D. Quartel, H. Jonkers, M. van Sinderen, Extending enterprise architecture modelling with business goals and requirements, Enterprise information systems 5 (1) (2011) 9–36.
- [84] N. Rurua, R. Eshuis, M. Razavian, Representing variability in enterprise architecture, Business & Information Systems Engineering 61 (2) (2019) 215–227.
- 1210 [85] M. La Rosa, W. M. van der Aalst, M. Dumas, A. H. Ter Hofstede, Questionnaire-based variability modeling for system configuration, Software & Systems Modeling 8 (2) (2009) 251–274.
- [86] M. Rosemann, W. M. Van der Aalst, A configurable reference modelling language, Information systems 32 (1) (2007) 1–23.
- 1215 [87] A. Hallerbach, T. Bauer, M. Reichert, Capturing variability in business process models: the provop approach, Journal of Software Maintenance and Evolution: Research and Practice 22 (6-7) (2010) 519–546.
- [88] K. Pohl, G. Böckle, F. Van Der Linden, Software product line engineering: foundations, principles, and techniques, Vol. 1, Springer, 2005.

- 1220 [89] L. Chen, M. A. Babar, A systematic review of evaluation of variability management approaches in software product lines, *Information and Software Technology* 53 (4) (2011) 344–362.
- [90] K. Schmid, I. John, A customizable approach to full lifecycle variability management, *Science of Computer Programming* 53 (3) (2004) 259–284.
- 1225 [91] M. Svahnberg, J. Van Gurp, J. Bosch, A taxonomy of variability realization techniques, *Software: Practice and experience* 35 (8) (2005) 705–754.
- [92] M. Sinnema, S. Deelstra, P. Hoekstra, The covamof derivation process, in: *International Conference on Software Reuse*, Springer, 2006, pp. 101–114.
- [93] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, A. Wasowski, Cool features and tough decisions: a comparison of variability modeling approaches, in: *Proceedings of the sixth international workshop on variability modeling of software-intensive systems*, 2012, pp. 173–182.
- 1230 [94] R. Sharpe, K. Van Lopik, A. Neal, P. Goodall, P. P. Conway, A. A. West, An industrial evaluation of an industry 4.0 reference architecture demonstrating the need for the inclusion of security and human components, *Computers in industry* 108 (2019) 37–44.
- 1235 [95] P. Avgeriou, Describing, instantiating and evaluating a reference architecture: A case study, *Enterprise Architecture Journal* 342 (2003) 1–24.
- [96] E. Cioroica, S. Chren, B. Buhnova, T. Kuhn, D. Dimitrov, Towards creation of a reference architecture for trust-based digital ecosystems, in: *Proceedings of the 13th European Conference on Software Architecture-Volume 2*, pp. 273–276.
- 1240 [97] R. Kazman, L. Bass, G. Abowd, M. Webb, Saam: A method for analyzing the properties of software architectures, in: *Proceedings of 16th International Conference on Software Engineering*, IEEE, 1994, pp. 81–90.
- 1245

- [98] P. Bengtsson, N. Lassing, J. Bosch, H. van Vliet, Architecture-level modifiability analysis (alma), *Journal of Systems and Software* 69 (1-2) (2004) 129–147.
- [99] L. G. Williams, C. U. Smith, Pasasm: a method for the performance assessment of software architectures, in: *Proceedings of the 3rd international workshop on Software and performance*, pp. 179–189.
- [100] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere, The architecture tradeoff analysis method, in: *Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems* (Cat. No. 98EX193), IEEE, pp. 68–78.
- [101] P. Bengtsson, J. Bosch, Scenario-based software architecture reengineering, in: *Proceedings. Fifth International Conference on Software Reuse* (Cat. No. 98TB100203), IEEE, 1998, pp. 308–317.
- [102] B. Graaf, H. Van Dijk, A. Van Deursen, Evaluating an embedded software reference architecture-industrial experience report, in: *Ninth European Conference on Software Maintenance and Reengineering*, IEEE, 2005, pp. 354–363.
- [103] A. J. Rohling, V. V. G. Neto, M. G. V. Ferreira, W. A. Dos Santos, E. Y. Nakagawa, A reference architecture for satellite control systems, *Innovations in Systems and Software Engineering* 15 (2) (2019) 139–153.
- [104] E. Y. Nakagawa, R. M. Martins, K. R. Felizardo, J. C. Maldonado, Towards a process to design aspect-oriented reference architectures, in: *XXXV Latin American Informatics Conference (CLEI) 2009*, 2009.
- [105] E. M. Leonard, Design and implementation of an enterprise data warehouse, Marquette University, 2011.
- [106] W. Brackenbury, R. Liu, M. Mondal, A. J. Elmore, B. Ur, K. Chard, M. J. Franklin, Draining the data swamp: A similarity-based approach,

- in: Proceedings of the workshop on human-in-the-loop data analytics, 2018, pp. 1–7.
- 1275 [107] J. Lin, The lambda and the kappa, IEEE Internet Computing 21 (05) (2017) 60–66.
- [108] Apache beam.
URL <https://beam.apache.org/>
- [109] Databricks.
1280 URL <https://databricks.com/>
- [110] Z. Deghani, How to move beyond a monolithic data lake to a distributed data mesh (2019).
URL <https://martinfowler.com/articles/data-monolith-to-mesh.html>
- 1285 [111] N. Alshuqayran, N. Ali, R. Evans, A systematic mapping study in microservice architecture, in: 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), IEEE, 2016, pp. 44–51.
- [112] B. O’Reilly, How to implement hypothesis-driven development (2014).
1290 URL <https://www.thoughtworks.com/insights/articles/how-implement-hypothesis-driven-development>
- [113] N. Ford, R. Parsons, P. Kua, Building evolutionary architectures: support constant change, ” O’Reilly Media, Inc.”, 2017.
- [114] S. Newman, Building microservices, ” O’Reilly Media, Inc.”, 2021.
- 1295 [115] A. Bellemare, Building Event-Driven Microservices: Leveraging Organizational Data at Scale, O’Reilly Media, 2020.
URL https://www.amazon.com/Building-Event-Driven-Microservices-Leveraging-Organization/dp/1492057894/ref=sr_1_1?crid=1WQYXJT85E0CL&keywords=event+

- driven+microservices&qid=1647139910&sprefix=event+driven+
 1300 microservi%2Caps%2C307&sr=8-1
- [116] E. Evans, E. J. Evans, Domain-driven design: tackling complexity in the heart of software, Addison-Wesley Professional, 2004.
- [117] L. Aceto, A. Ingólfssdóttir, K. G. Larsen, J. Srba, Reactive systems: modelling, specification and verification, cambridge university press, 2007.
- 1305 [118] Z. Dehghani, Data mesh principles and logical architecture (2020).
 URL <https://martinfowler.com/articles/data-mesh-principles.html>
- [119] P. Di Francesco, Architecting microservices, in: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), IEEE, 2017,
 1310 pp. 224–229.
- [120] O. Zimmermann, Microservices tenets, Computer Science-Research and Development 32 (3) (2017) 301–310.
- [121] M. Waseem, P. Liang, M. Shahin, A systematic mapping study on microservices architecture in devops, Journal of Systems and Software 170
 1315 (2020) 110798.
- [122] Terraform is an open-source infrastructure as code software tool that provides a consistent cli.
 URL <https://www.terraform.io/>
- [123] The leading customer data platform.
 1320 URL <https://segment.com/>
- [124] The hadoop distributed file system.
 URL https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
- [125] The package manager for kubernetes.
 URL <https://helm.sh/>

- 1325 [126] Gnu make.
URL [https://www.gnu.org/software/make/manual/make.html#
toc-An-Introduction-to-Makefiles](https://www.gnu.org/software/make/manual/make.html#toc-An-Introduction-to-Makefiles)
- [127] A. Josey, TOGAF® Version 9.1-A Pocket Guide, Van Haren, 2016.